Case Study: Modernize a Food Delivery Service with Cloud

Estimated time needed: 30 minutes

DineEase

DineEase is a food ordering system that offers a mobile application for end-users (consumers) and a web application for back-office and administrative users. Consumers can order food from any number of restaurants. The system locates the consumer's selected restaurant, sends the order to the restaurant, collects money from the consumer, and disburses the consumers payments to the restaurant and the delivery person.

Your mission for this project is to assist with moving the legacy DineEase platform, built on top of a monolith architecture, to a Cloud Native architecture built on microservices. After evaluating several Cloud providers, DineEase chose IBM Cloud to deploy the revamped platform. As part of the project, you will become familiar with the current platform components. Then, you will provide recommendations for the IBM Cloud services that are most suitable for DineEases' new Cloud Native deployment.

DineEase platform components

DineEase has several components (modules) including:

Menu

A module serves the menu details of a restaurant. This module interacts with the database to access, retrieve, and manage information about menu items and their categories. Common functions include:

- Get Menu Items: Retrieve a list of all items available on the menu, including their names, descriptions, prices, and images.
- Get Menu Categories: Retrieve a list of menu categories (such as appetizers, main courses, desserts) to organize food items.
- Filter by Category: Retrieve menu items specific to a particular category.

Pricing

A module provides dynamic pricing based on customer location, previous order history, public holidays, weekends, or an event. It provides a centralized platform to manage and update pricing for all menu items and services. This pricing module allows restaurant owners to make changes in one place and have those updates reflected across all connected systems in real-time.

Ordering

A module that accepts orders placed by customers. This module is a critical component that handles taking and managing customer orders. The module enables restaurants to efficiently manage incoming orders, ensure accurate preparation and delivery, and provide a seamless ordering experience for consumers.

Payment

A module takes pricing and ordering details and charges the payment method. And the process all happens securely, building consumer confidence. This module accepts multiple payment methods and seamlessly integrates with Point of Sales (POS) systems.

Dispatch

A module that passes the customer order details to restaurants so restaurants can start cooking and arrange delivery to the customer. Meanwhile, align the delivery partner, ensuring that customers are not waiting too long before the food is ready and that delivery people are not turning up too late after the food is ready. This module also provides real time tracking, so the consumers know when their food will be at their door.

DineEase database

All of the DineEase data is stored in self-managed relational database. Complex relationships are defined between entities.

Relational database

A relational database is a database management system (DBMS) that stores and organizes data in a structured format using tables, rows, and columns. Relational databases are based on the relational model, introduced by Dr. E.F. Codd in 1970. In a relational database, data is represented in the form of tables, and the relationships between different tables are established using keys.

The relational model provides several advantages, including data integrity, data consistency, and the ability to perform complex queries using Structured Query Language (SQL).

For example, a restaurant has a menu with multiple items listed. Each menu item has an associated price, and the database must store multiple rows to allow changes based on the price list.

Key components of a relational database include:

- Table: A table is a collection of related data organized into rows and columns. Each row in the table represents a single record or data entry, while each column represents a specific attribute or field for that record.
- Row: A row represents a single instance of data in a table. A row contains values for each column, representing the attributes of that particular record.
- Column: Also known as a field, a column represents a specific attribute or data type shared by all rows in the table. Each column has a unique name that identifies its purpose.
- Relationships: Relationships in a relational database define how data in different tables are related or connected. Hence the name, relational database.

Monolith challenges

DineEase was built as a monolith application and deployed on servers manually. But the recent demand has highlighted some challenges with that architecture. Your task is to make the application Cloud-ready.

- Scalability: Unable to scale certain parts of the platform. For example, the Ordering service on its own can not be scaled to meet demands in certain countries on special events. For example Super Bowl in the US, Olympic Games ceremony.

- Low latency and high availability: Dispatch service depends on regional players but with monolith design, it is difficult to target high availability without too much overhead.
- Dedicated workload: The Payments part of the architecture requires real-time processing and computation of fraud checks. But it cannot be scaled on its own.
- Only API: Search, Menu and Pricing can be converted to API services, so that these services can be accessed from different channels.
- Improve read performance: Before placing an order, customers like to see a restaurant\s latest ratings and reviews. Joining data from Restaurant, Review and Rating tables is proving to be difficult with the growth in number of restaurants and users.
- Regional databases: Having a single self-managed global database is proving to be difficult and creates unwanted limitations.

DineEase 2.0

What are Microservices?

Microservices (or microservices architecture) is a cloud-native architectural approach in which a single application is composed of many loosely coupled and independently deployable smaller components or services. These services typically:

- Have their own technology stack, inclusive of the database and data management model;
- Communicate with one another over a combination of REST APIs, event streaming, and message brokers; and
- Are organized by business capability, with the line separating services referred to as a bounded context.

Read more about Microservices.

1. Provide low latency and high availability

DineEase wants to benefit from regional deployment of services; so it can provide low latency access to delivery partners for dispatch services. These services provide real time updates of orders to both customer and delivery partner, tell delivery partners when it's ready to be collected, and show journey and estimated time of arrival (ETA) to the customer.

The availability requirement for these services is 99.99%, and they should use separate physical locations within a region to prevent failures on data center outages.

Task 1: Visit IBM Cloud Regions and identify which regional deployment setup will suit this need.

2. Run dedicated workloads

Payment processing for DineEase is computationally intensive, needs a secure environment, and has to process large amounts of data very quickly, so there are no delays in payment processing and fraud checks.

For this kind of workload, you need services that provide direct access to the underlying physical hardware, eliminate the overhead associated with virtualization, and have dedicated access to the entire physical server without sharing any RAM or CPU resources with other tenants and can also provide a graphics processing unit (GPU) to power the performance where the CPU lacks power.

Task 2: Visit Compute solutions on IBM Cloud and identify which compute service offers the capabilities needed for running a dedicated workload as mentioned above.

3. Use API services

Search, Menu, and Pricing services for DineEase need the flexibility of using different technology stacks per service, avoiding the hassle of setting up individual servers for each service. These services must have individual development and deployment life cycles and be able to run on hosts with different operating systems.

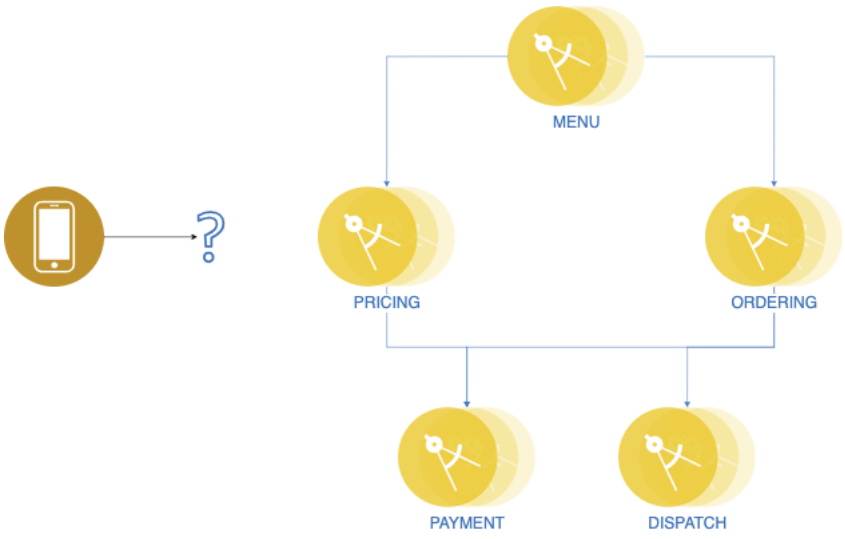
These services will run code in response to HTTP-based API requests from mobile applications, third-party services, or from interacting with each other, and these services must be lightweight and portable.

Task 3: Visit Containers on IBM Cloud and identify computer services that offer the capabilities needed for running a flexible set of services that can scale automatically and independently and are self-healing.

To also provide the capability to interact among these services or from the outside world and remove rigid dependency on each other, you need to add a layer of communication in between.

This component should also provide request routing, which consults a routing map that shows where each request should be sent.

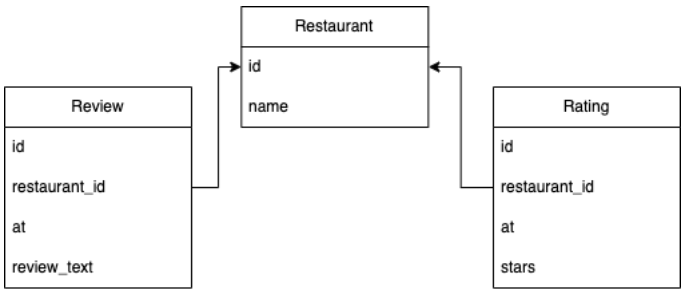
4. Identify the right gateway



Task 4: Visit Gateways on IBM Cloud and identify the right type of gateway needed to provide this layer.

5. Improve read performance

Providing optimized reads to let consumers see reviews and ratings of a restaurant has been challenging. Because the platform is growing, this is becoming difficult to maintain due to the growth in data sizes.



To support this growth, developers decided to bring in a document database to enhance read performance, and avoid complex joins on relational databases, but still keep the capability of complex queries for Reviews of a restaurant.

The schema of this temporal data must be flexible to accommodate unstructured data.

Use document-based databases

A document-based database is a type of NoSQL (Not Only SQL) database that stores and manages data in the form of documents. In this context, a document refers to a self-contained, hierarchical data structure typically represented in formats like JSON (JavaScript Object Notation) or BSON (Binary JSON). These documents can vary in structure and can have nested fields, making them flexible for storing different types of data.

Unlike traditional relational databases, which store data in tables with predefined schemas, document-based databases allow for schema flexibility.

```
restaurant = {
    id: "Yb5xwX",
    name: "Flavors & Tales",
    latestReviews: [
        { at:ISODate("2023-07-01T16:22:01.090Z"), text: "Wow! F&T has won my heart with their mouthwatering dishes." },
        { at:ISODate("2023-06-24T21:02:54.989Z"), text: "The Smoky Ribs were fall-off-the-bone tender, and the BBQ sauce was finger-licking good
    ],
    starRatings:{
        totalRatings: 34,
        stars: [9, 20, 5, 0, 0],
        average: 4.1
    }
}
```

Task 5: Visit the IBM Cloud database options and identify a document-based database that deploys and scales instantly to meet requirements and provides built-in offline-first architecture.

6. Ease transactional database administration

So far DineEase has managed to run its ordering system on a self-managed relational database to satisfy its data demands. But this requires too much operational overhead to administer the database and is proving difficult every time additional capacity needs to be added. Now you are ready to harness the Cloud by using a managed database service that can automatically scale capacity.

The key requirements are:

- The new service has to be fully managed by the provider
- Should provide high-availability disaster recovery (HADR)
- Still a relational database
- Delivers fast query processing with enterprise-level performance
- Capabilities for online transactional processing

Task 6: Visit IBM Cloud Database and identify a relational database that satisfies the preceding criteria.

7. Distribute media quickly

DineEase hosts a lot of food photos, restaurant logos, and other secondary media. Currently, these photos are served by the same web host that serves the monolith web application. After the new design moves away from the monolith, you will still need to provide a mechanism that hosts this media. The tool that provides such capability helps with reducing latency by distributing across multiple geographic locations, bringing content closer to consumers, reducing the workload on the web server by hosting static content and saving bandwidth by leveraging compression and caching. The tool can also adapt the content to the consumer's device type.

Task 7: Visit IBM Cloud Catalog Networking products and find the service that distributes your content in geographically diverse nodes, shortens the distance the content must travel to get to your end user, takes advantage of superior web and mobile performance, and provides content delivery solutions.

Operations 2.0

8. Increase uptime and availability

Now that each functional part of DineEase is divided into dedicated microservices, next consider how the load can be distributed amongst multiple instances. This service should provide us with high availability by distributing incoming traffic across multiple instances evenly, can provide fault tolerance when it detects failures, and provides a central point to access a "type" of resource, for example, menu service.

Task 8: Visit Cloud Networking Solutions and find a service that you can use to balance traffic among your servers to help improve uptime. And can easily scale your applications by adding or removing servers with minimal disruption to your traffic flows.

9. Perform health monitoring

With the introduction of new cloud technologies, you need to be equipped with the right tools and solutions to keep operations running.

Task 9: Visit Monitoring on IBM Cloud and locate a cloud-native management system you can include as part of your IBM Cloud architecture that provides you with operational visibility into the performance and health of your applications, services, and platforms. This system must also support monitoring and troubleshooting, the definition of alerts, and the creation of custom dashboards.

Conclusion

Congratulations, you defined the key components needed for the DineEase architecture in the Cloud!

Author(s)

Muhammad Yahya